Sissejuhatus infotehnoloogiasse

6.B loeng: andmetüübid

Loengu sisu

Mis on andmetüübid

Lihtsad andmetüübid

- Täisarv
- Ujukoma-arv
- Sümbol ehk täht

Kompleksandmetüübid

- String ehk tekst
- Massiiv
- List
- Puu
- Hash

Mis asjad on andmetüübid?

Levinud viisid andmeid arvutis kodeerida.

Protsessor toetab otse ainult lihtandmetüüpe: täisarve ja ujukoma-arve. Tõeväärtused ja tähed on täisarvu erivariandid. Mäluaadress ehk pointer on samuti lihtsalt täisarv.

Kompleksandmetüübid on praktilised viisid, kuidas panna kokku hulga andmeid (tekstid, massiivid, loendid, dictid jne).

Kompleksandmetüüpide mugavaks ehitamiseks ja kasutamiseks on enamuses programmeerimiskeeltes hulga

- Spetsiaalseid teegifunktsioone.
- Spetsiaalset süntaksit
- Aga protsessori jaoks nad ei ole „asjad“ mida protsessor otse toetaks.

Igaüks võib leiutada juurde oma andmetüüpe.

Täisarvud

A la 34, -4545, 0, 92829292

Põhiasjad, millega protsessor tegutseda oskab.

Protsessor oskab täisarvudega teha:

- Mälust lugeda ja mällu kirjutada
- Aritmeetikatehteid
- Kasutada neid mälupesade aadressidena, kust lugeda või kuhu kirjutada.

Täisarvud

Täisarvude suurusel ei ole iseenesest ranget piiri ees.

Aga:

- Protsessor oskab mäluaadressidena kasutada 4-baidiseid ja 8-baidiseid täisarve
- Protsessor oskab aritmeetikat teha 4-baidiste ja 8-baidiste täisarvudega
- Cache loetakse mälust alati korraga vähemalt 8 baiti

Seega enamasti kasutataksegi kas 4-baidised või 8-baidiseid täisarve, kuigi saab kasutada ka 2-baidiseid, 16-baidiseid jne.

4-baidine ilma märgita: 0 ... 4,294,967,295 ehk $2^{32} - 1$

4-baidine märgiga: −2,147,483,648 ... 2,147,483,647

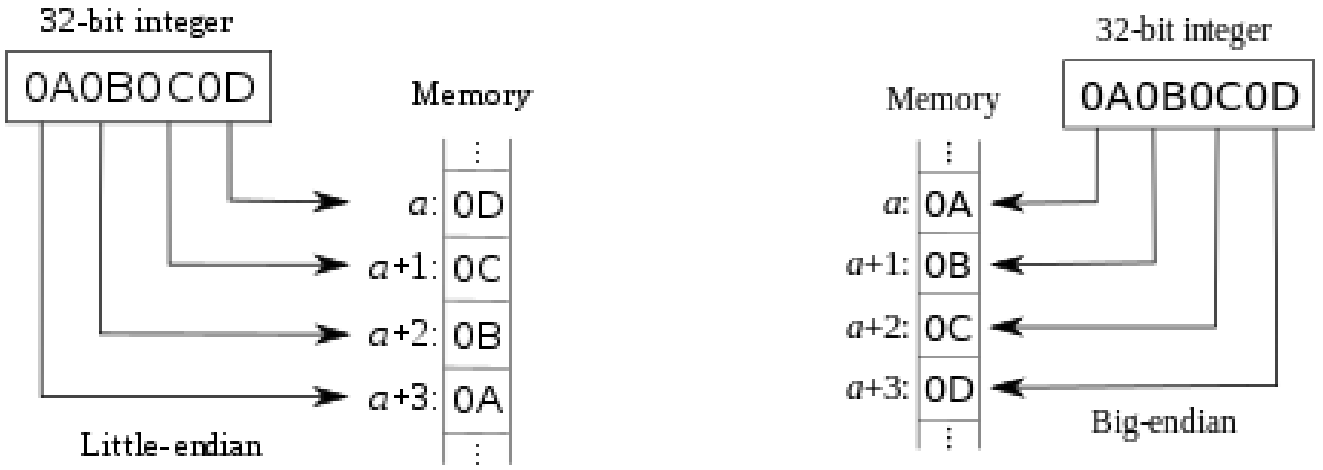
Negatiivsed täisarvud

Esimene bitt määrab, kas arv on negatiivne või ei. Vaatame ühebaidiseid arve nii märgita kui märgiga tähenduses, liites järjest ühe:

0000 0000 on 0
0000 0001 on 1
....
0010 0011 on 35
...
0111 1111 on 127
1000 0000 on 128 või **-128**
1000 0001 on 129 või **-127**
...
1111 1110 on 254 või **-2**
1111 1111 on 255 või **-1**

Little and big endian

Millist pidi hoida mälus täisarve? Inimesed kirjutavad **big-endian** viisil.



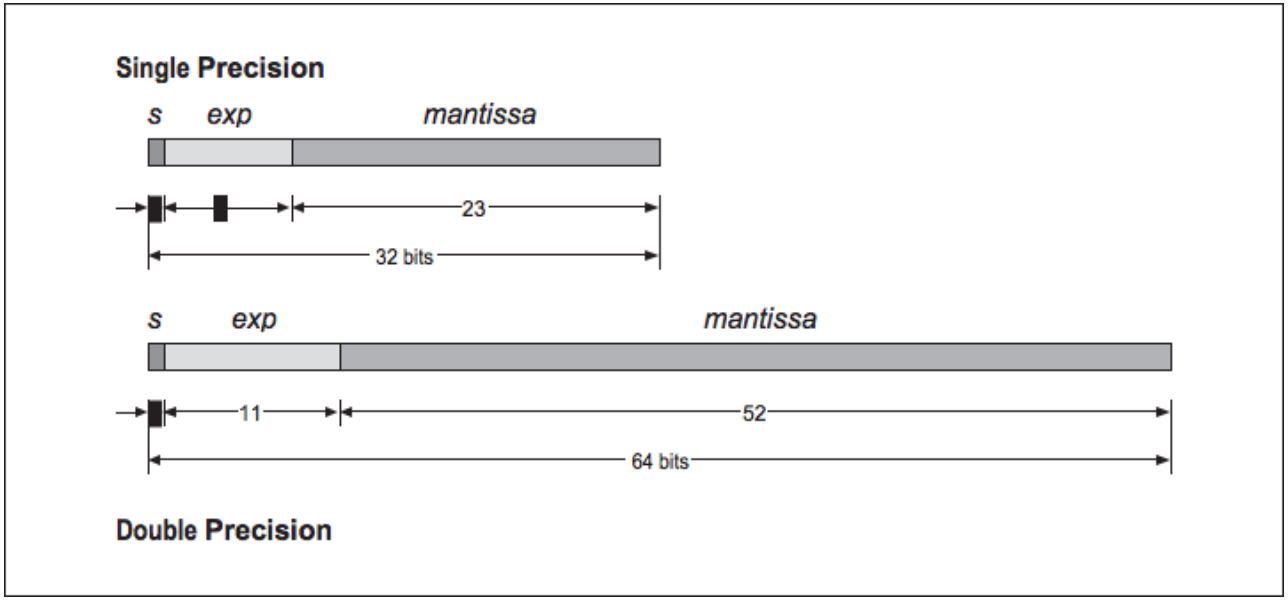
Enamik protsessoreid kasutab **little-endian** põhimõtet

Ujukoma-arvud

Kuidas kodeerida komaga (ja väikseid ja suuri!) arve nagu

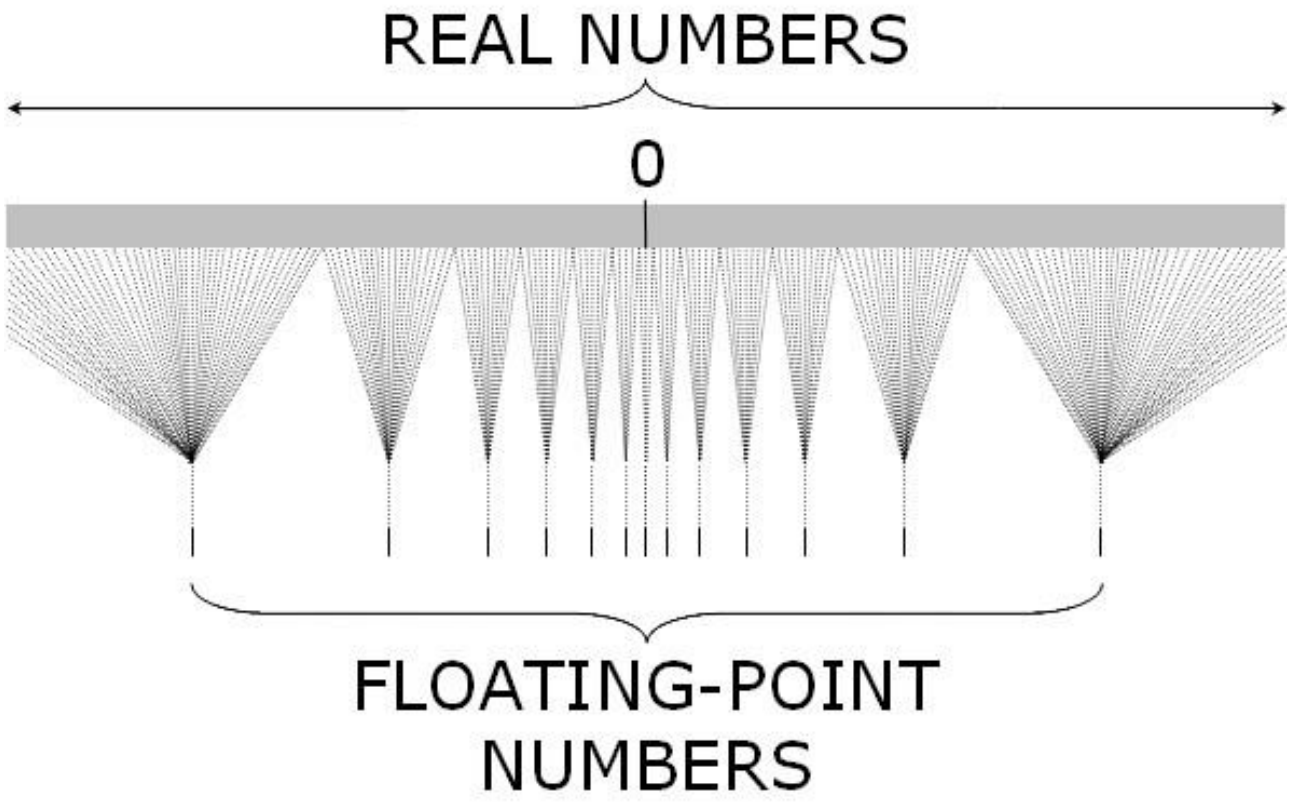
1.34566
0.0122
0.0000000000000001
10000000000000000000

Tüüpiliselt kasutatakse 8 baiti. Teine variant on 4 baiti.
Levinud kodeering nn IEEE standard.



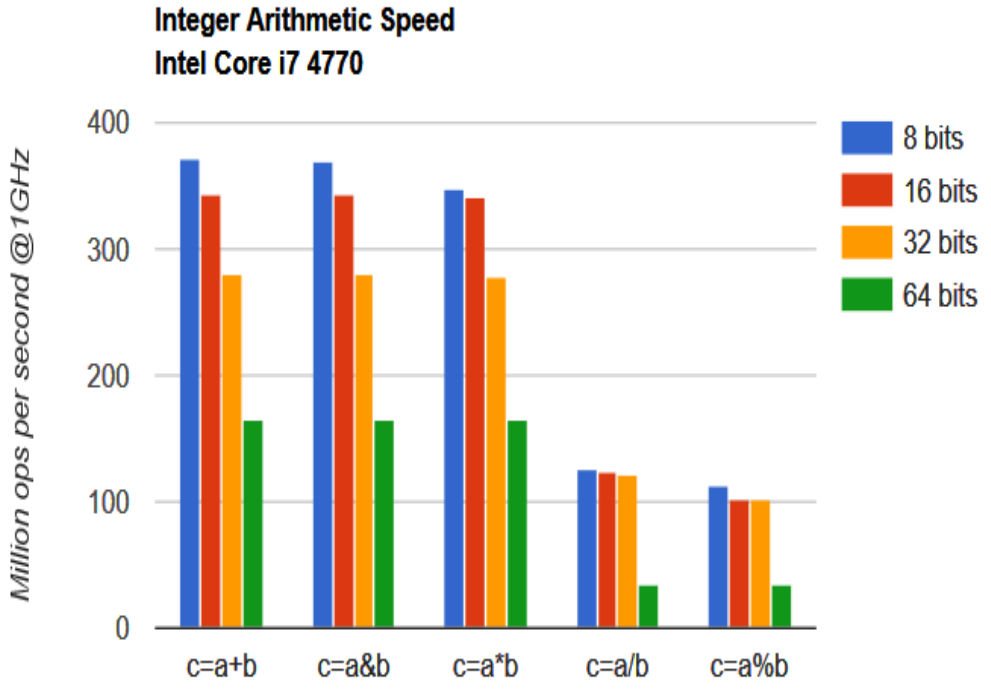
Ujukoma-arvud

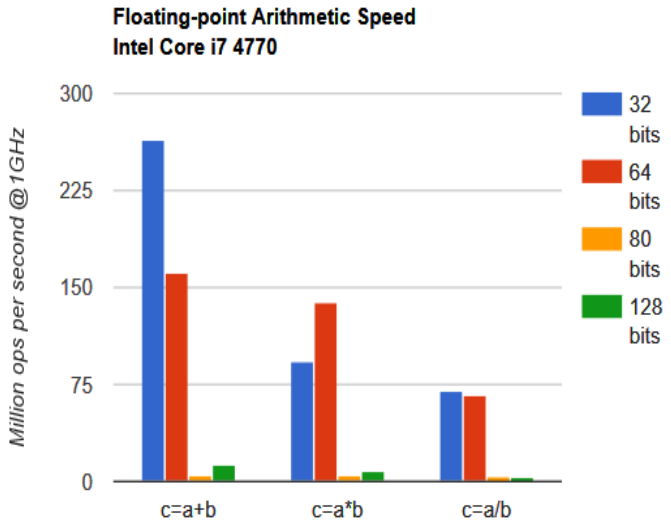
„Mantissa“ on sisuliselt offset „eksponendiga“ määratud piirkonnas. Ujukoma-arvud katavad seda hõredamalt reaalsete arvude hulka, mida suuremad arvud on, ehk, mida suurem eksponent.



Protsessori kiirus eri arvutüüpide jaoks

Protsessorid suudavad teha otse aritmeetikatehteid ujukoma-arvudega, aga need tehted on veidi aeglasemad, kui täisarvudega.





Tõeväärtus arvutis?

Ei ole mingit standardkodeeringut: eri progekeeled kasutavad tõeväärtuse kodeerimiseks erinevad väärtusi.

C keeles on arv 0 „vale“ ja kõik muud väärtused on „tõesed“.

Javascriptis on vale 0, ““, false, [], {} ehk tühi string, tühi massiiv, tühi objekt: kõik muu on „tõene“.

Üksikud tähed arvutis?

Osa progekeeli kasutab üksiku tähe kodeerimiseks ühte baiti (C).

Osa kasutab kahte baiti (Java).

Mõned kasutavad veel rohkem baite.

Aga, üksikute tähtede asemel kasutavad programmeerijaid tihti harilikke täisarve. Üksikutest tähtedest olulisem on pika teksti kodeering.

Teksti (stringi) kodeerimine arvutis?

Tekst on lihtsalt jada baite mälus järjest pluss pikkuse määrang.

Lihtsamal juhul (ascii ja iso-8859-1 kodeeringud) on üks täht üks bait.

Keerukamal juhul (utf-8) on ascii tähed üks bait, teised rohkem.

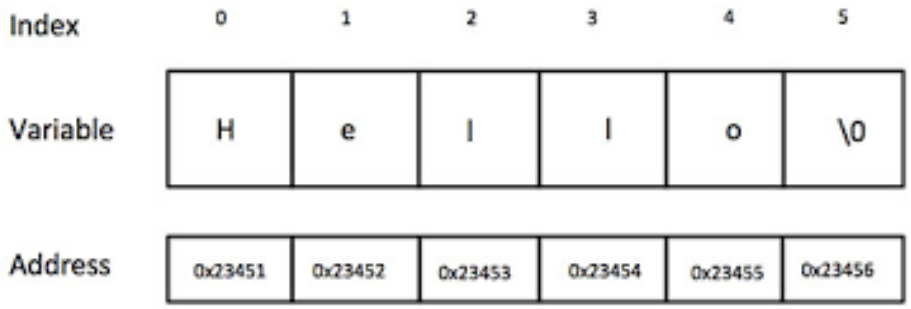
Java stringides on üks täht kaks baiti.

Pikkuse määrangut tehakse kahel eri viisil:

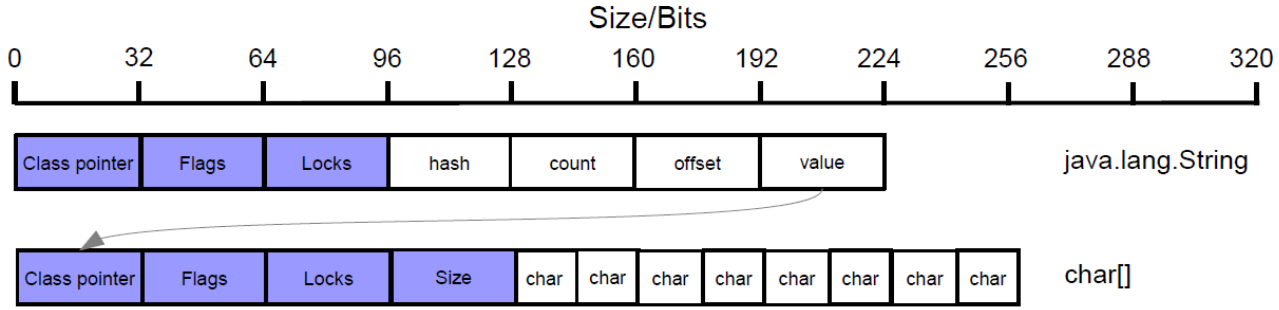
- C stringid lõpevad baidiga 0.
- Pythoni, Java, Javascripti jne stringide osaks on eraldi pikkuse number.

Teksti (stringi) kodeerimine arvutis?

- C stringid lõpevad baidiga 0.



- Java stringidel on pikkuse number ja veel lisainfot:
  - Simple example: java.lang.String containing “MyString”:

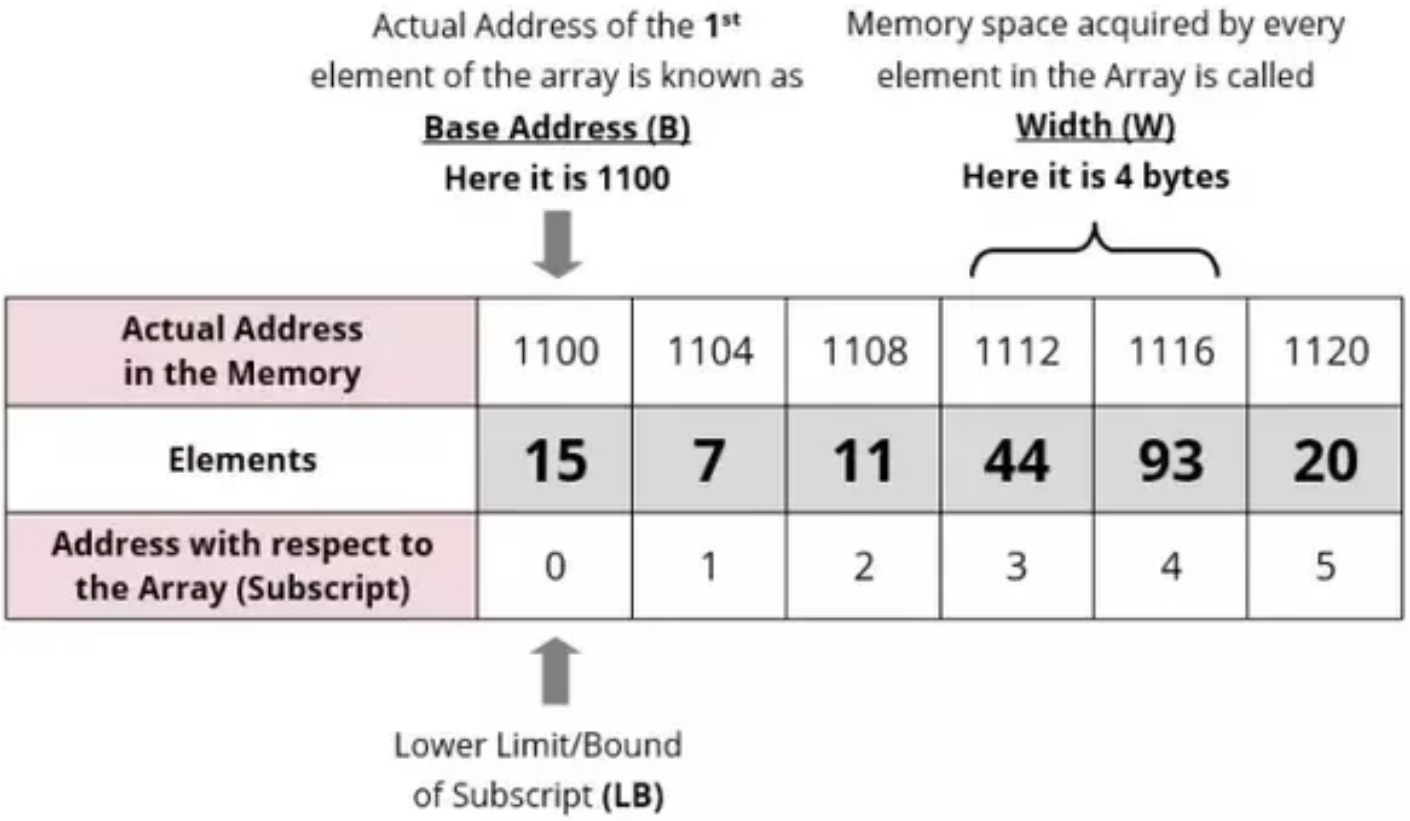


  - 128 bits of char data, stored in 480 bits of memory, size ratio of 3.75 : 1
    - Maximum overhead would be 24:1 for a single character!

Massiivid arvutis

Massiiv on jada ühetüübilisi väärtusi: tähti, täisarve, ujukoma-arve, teisi massiive vms.

Tüüpiline juhtum on lihtsalt jada täisarve mälus. Näide kahebaidiste arvude massiivist mälus:



Kui pikk on massiiv?

C keeles ei sisalda massiiv mingit pikkuse-numbrit. Programmeerija peab ise teadma või eraldi salvestama, et kui suur mõni tema tehtud massiiv on.

Java ning Javascripti ja Go massiivid sisaldavad lisaks andmetele ka massiivi pikkuse numbrit.

Kahemõõtmeline massiiv (tabel)

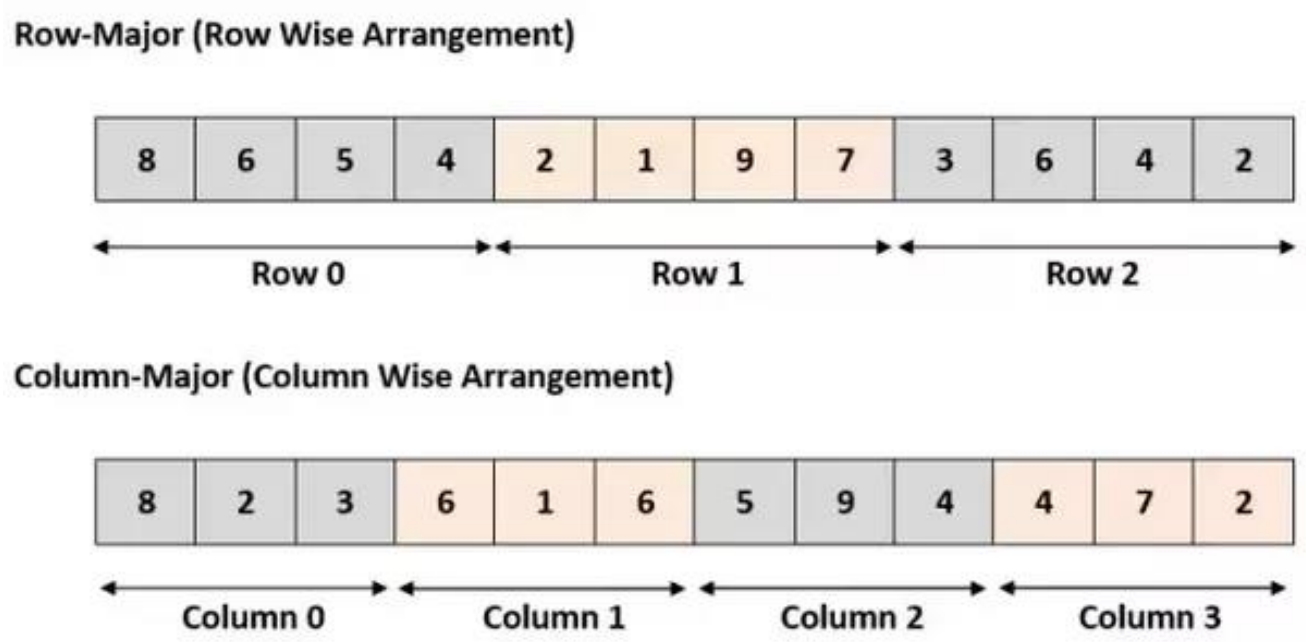
Olgu meil tabel

8 6 5 4

2 1 9 7

3 6 4 2

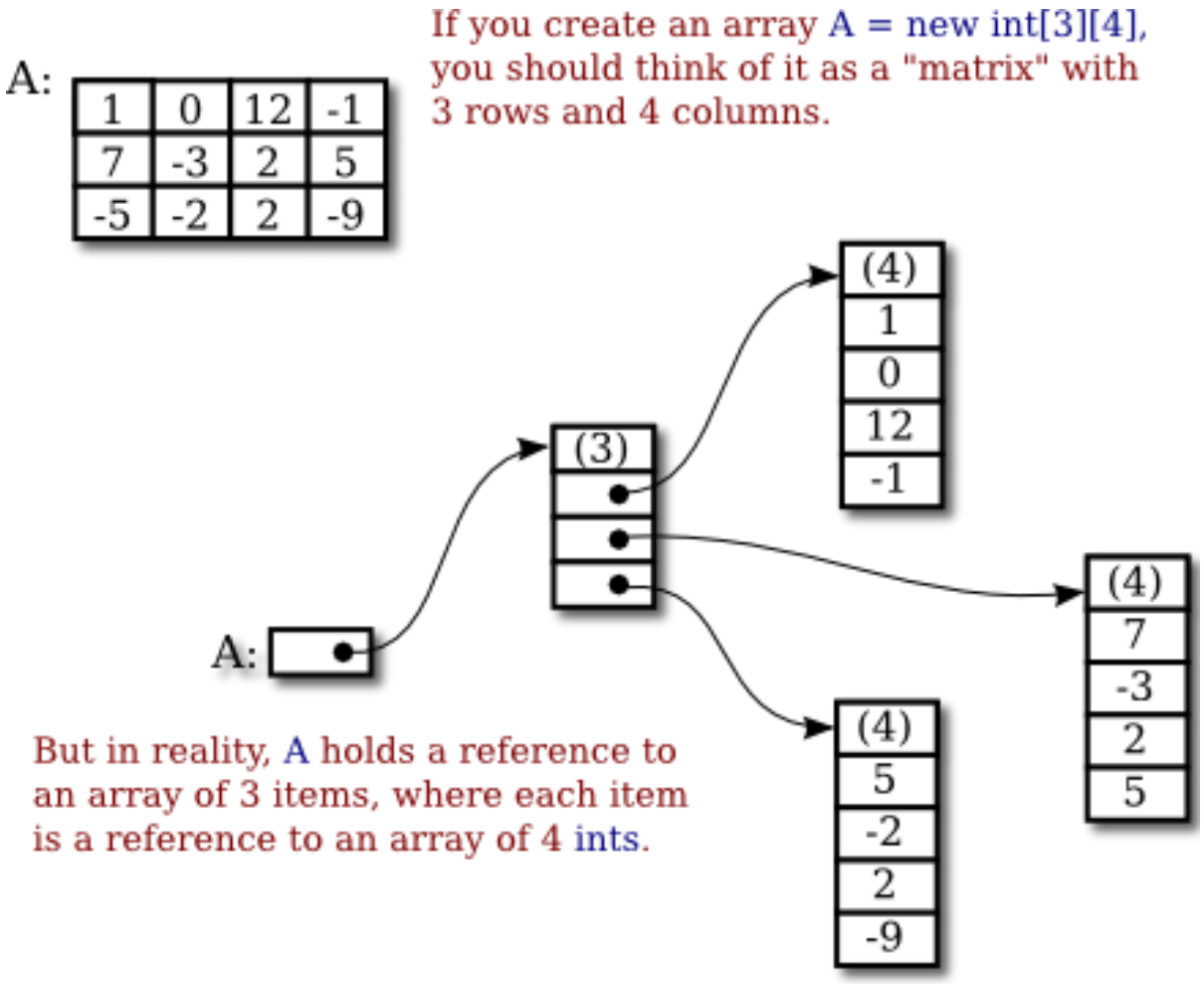
Kaks eri viisi seda hoida arvujadadena mälus:



Veel kolmas viis kahemõõtmelist massiivi hoida

Tulpade-massiiv sisaldab rea-massiivide alguse **mäluaadresse**.

Rea-massiivid ise ei pea olema lähestikku mälus.



Vahelduseks stringide massiiv

Olgu meil vahelduseks stringide massiiv:

pstr[0] = “Robert Redford“;

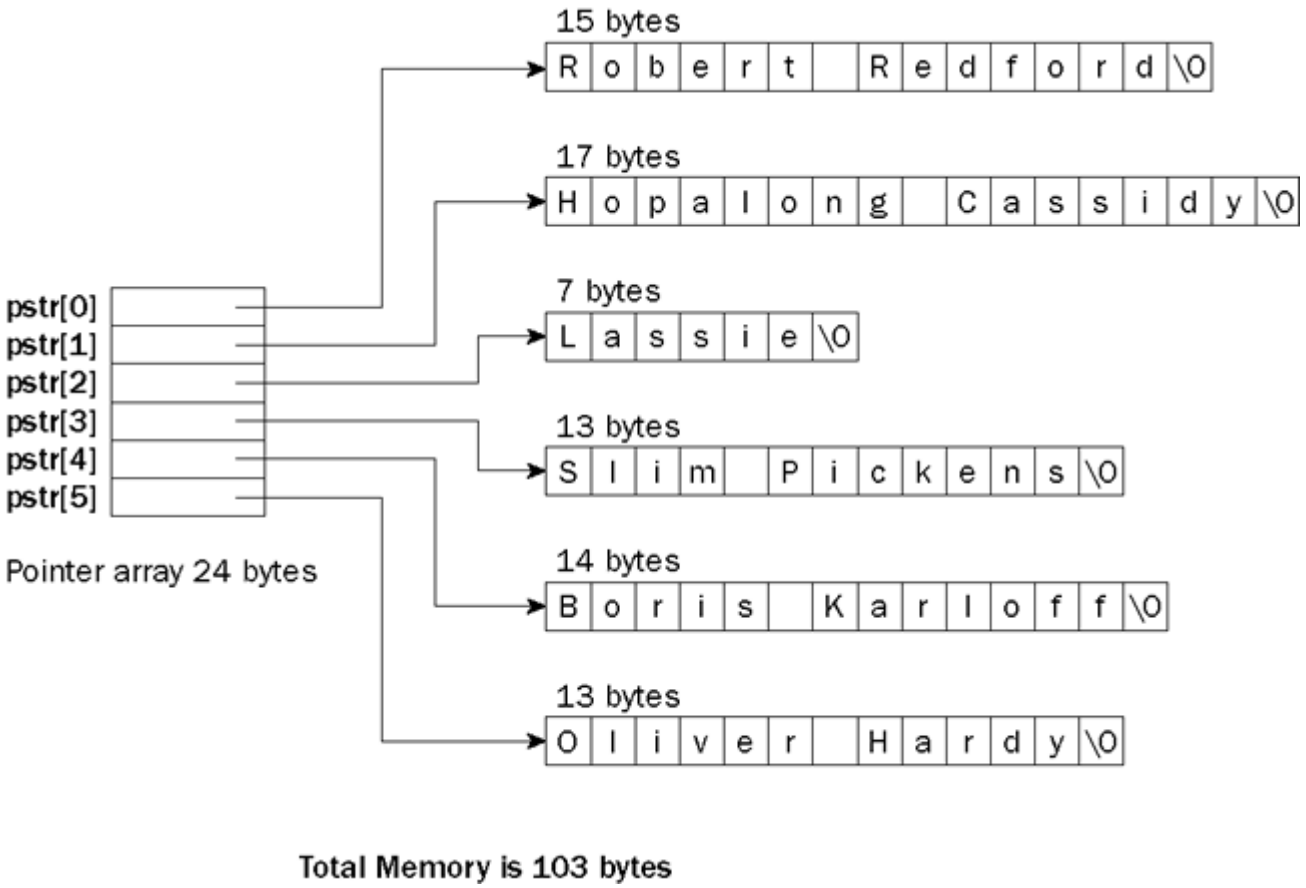
pstr[1] = “Hopalong Cassidy“;

pstr[2] = “Lassie“;

...

Vahelduseks stringide massiiv

Olgu meil vahelduseks stringide massiiv: ta sisaldab eri stringide alguspunktide **mäluaadresse**. Stringid ise ei pea olema lähestikku mälus.



Stringid võivad mälus olla ühised või erinevad!

Vasakul on kolm muutujat, mille väärtuseks sama string. Kui ühe muudad, „muutuvad“ ka teised kaks:

s1 = “Hello“; // s1, s2, s3 sisaldavad “Hello“ alguse mäluaadressi.

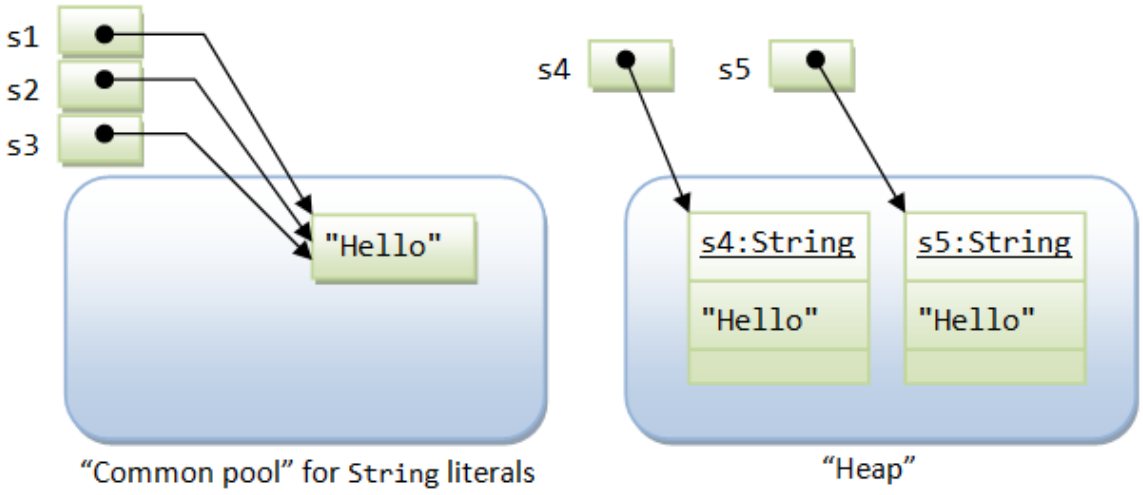
s2 = s1;

s3 = s1;

Paremal on kaks eraldi koopiat:

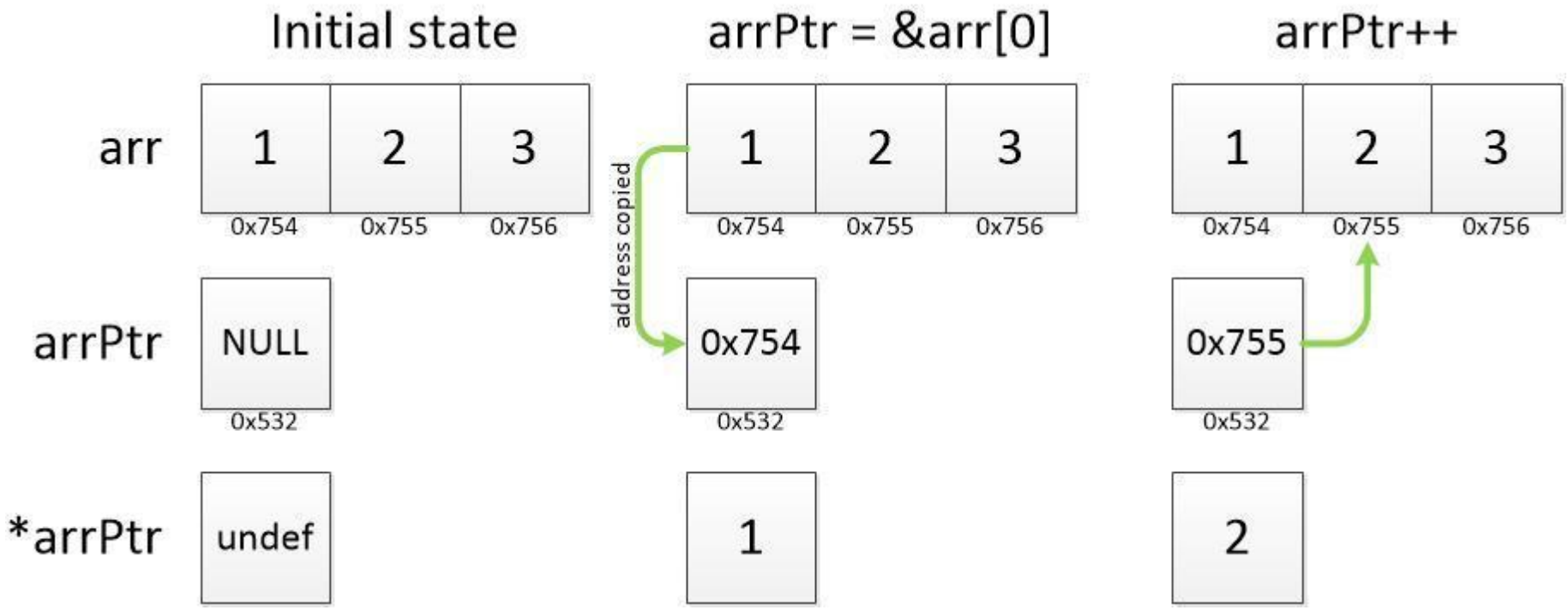
s4 =“Hello“;

s5 =“Hello“;



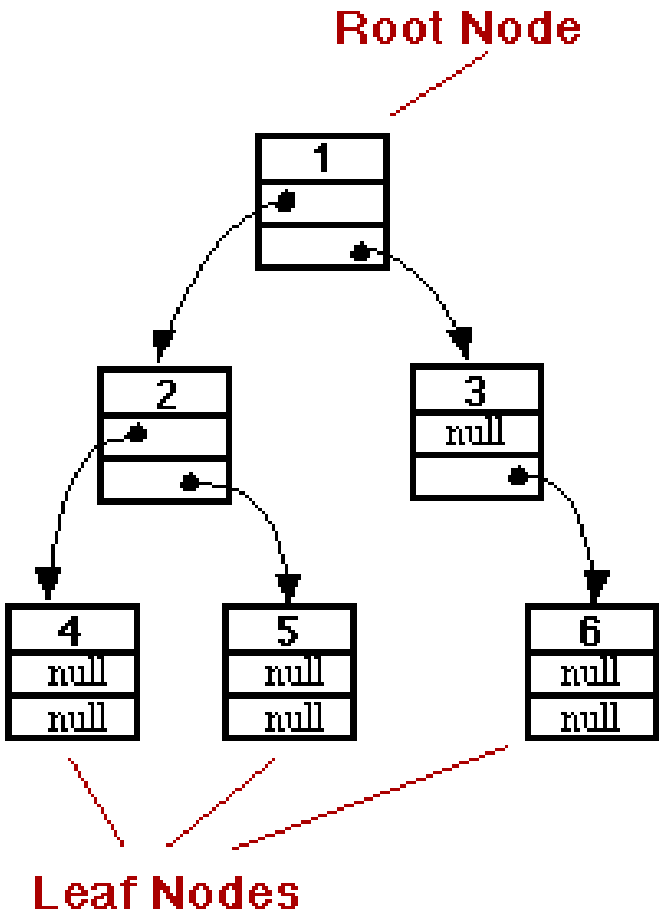
Pointeriaritmeetika C keeles

Pointerid ehk mäluaadressid on samas harilikud numbrid.



Puud!

Siin näitepuus koosneb iga element väärtusest ja kahest mäluaadressist. Null tähistab aadressi 0, mida normaalselt aadressina ei kasutata.



Listid!

Mida mõeldakse, kui kirjutatakse näiteks

A = [1, 5, -2, 7];

Kas A on massiiv? Või list? Mis on list?

Mis on näiteks

B = [[1, [5, -2], 7], 10, 20, “tere“, [2017], “tali“];

See on nn nested list, ehk listid üksteise sees mistahes kujul.

Listi erinevus massiivist

Massiiv on hulk kõrvuti sama tüüpi või sama suurusega objekte (täisarvud, ujukoma-arvud alammassiivid vms)

List üldiselt tähendab, et objektid ei pea olema sama tüüpi/sama suured ja nad võivad sisaldada kuitahes sügavaid alamliste.

NB! Kui sa Pythonis kirjutad

A = [1, 5, -2, 4]

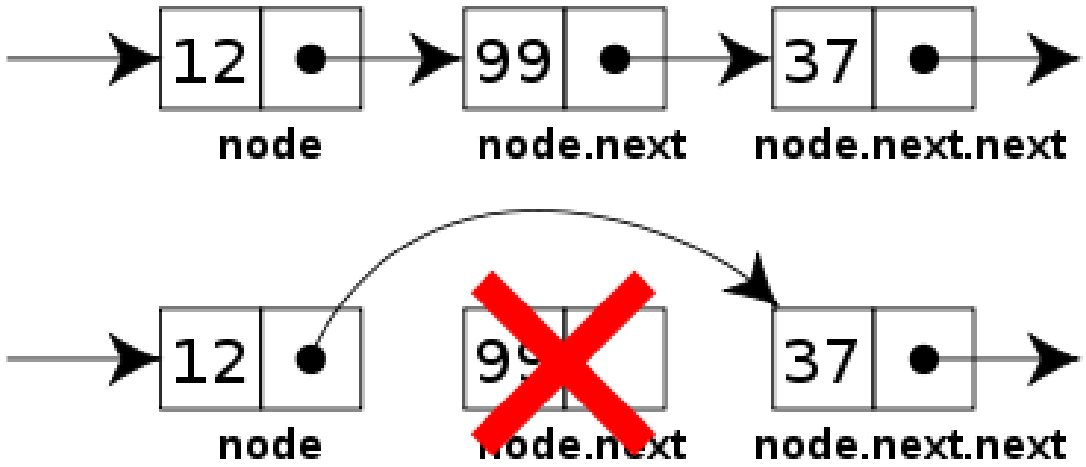
siis tehakse sulle massiiv mäluaadressidest: pointeritest karbistatud objektidele.

Lihtne list ja temast elemendi kustutamine

a = [12, 99, 37]

Ja siis tahaks 99 listist kustutada.

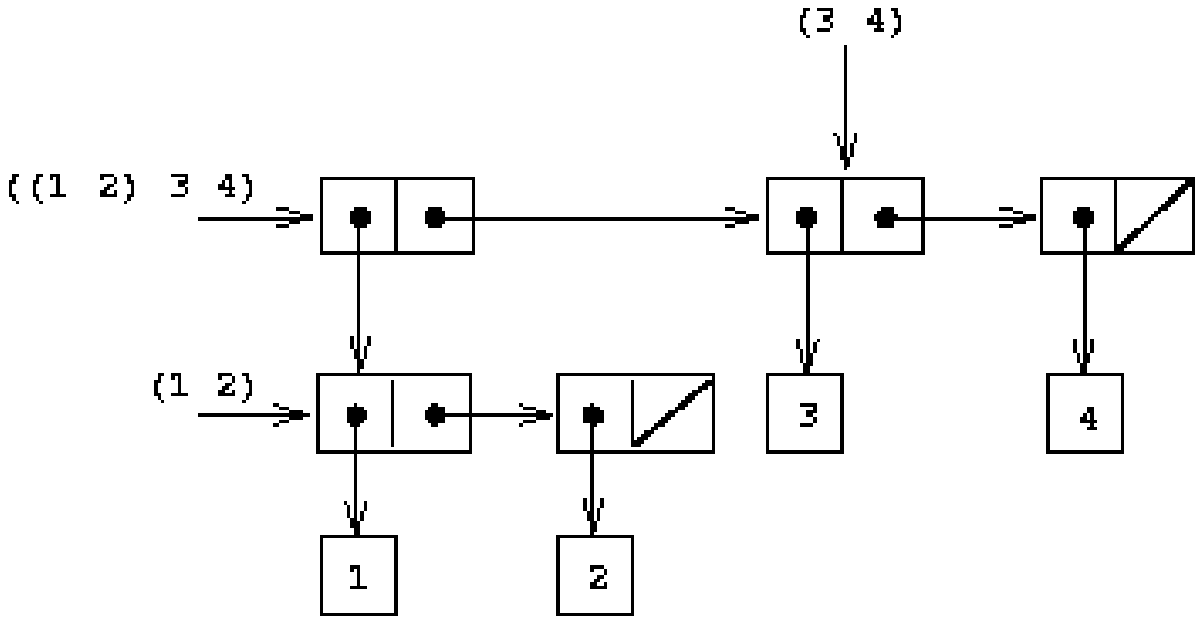
Mis sellest „99“ nodest edasi saab?



Listi esitus kahendpuuna

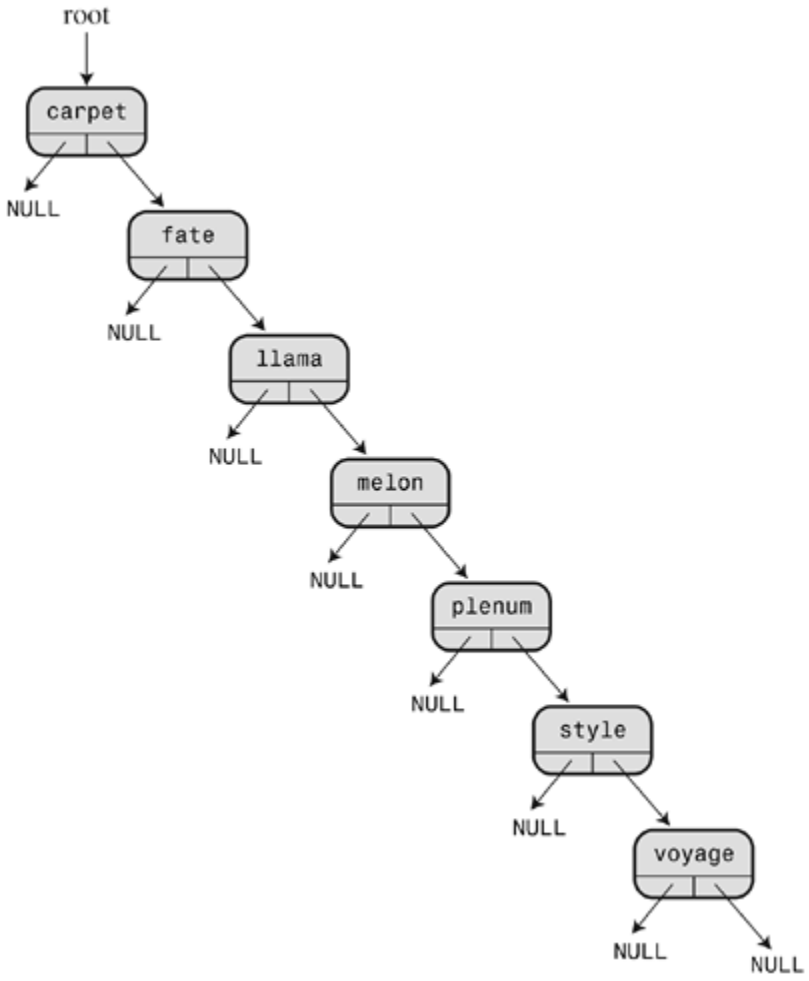
Olgu meil list [[1, 2], 3, 4] ehk Lispi süntaksis ((1 2) 3 4).

Kaldkriipsuga kast on mäluaadress (pointer) 0.



Listi esitus puuna

Lame list [“carpet“, “fate“, ... , “voyage“] on lihtsalt väga ühepoolne puu:

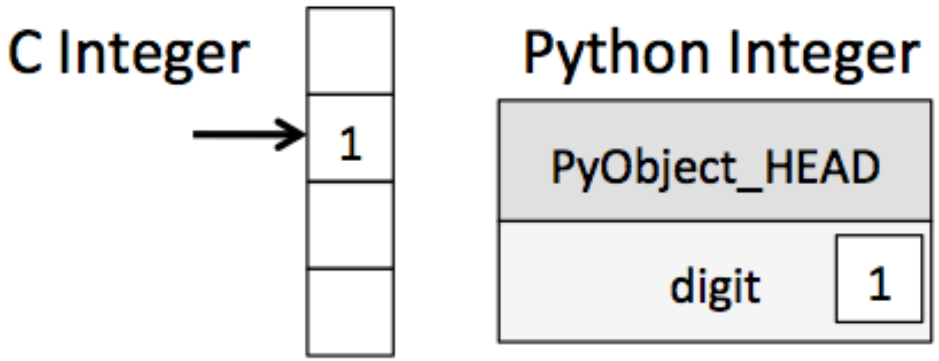


„Boxed“ väärtused

Kuidas me aru saame, et mingi täisarv mälus on kasutusel täisarvu, mitte pointeri või stringijupina vms?

C keele vastus: programmeerija võibki kasutada mõlemit moodi. Mäluaadressid ongi harilikud täisarvud. Ise teab, mis teeb.

Pythoni, Javascripti jne vastus: täisarv pannakse „karpi“ kus on lisainfo, et tegu on täisarvuga.



Vahemärkus: muutujate tüübid eri keeltes

C keeles on ok teha nii:

```
int x;                  // x on täisarvu tüüpi muutuja, mille jaoks võetakse neli baiti mälu
x = 34;                 // x väärtus on otsene neljabaidine arv 34 sealsamas mälus
x = “John Smith“;  // x väärtus on stringi mäluaadress, ka neli baiti!
```

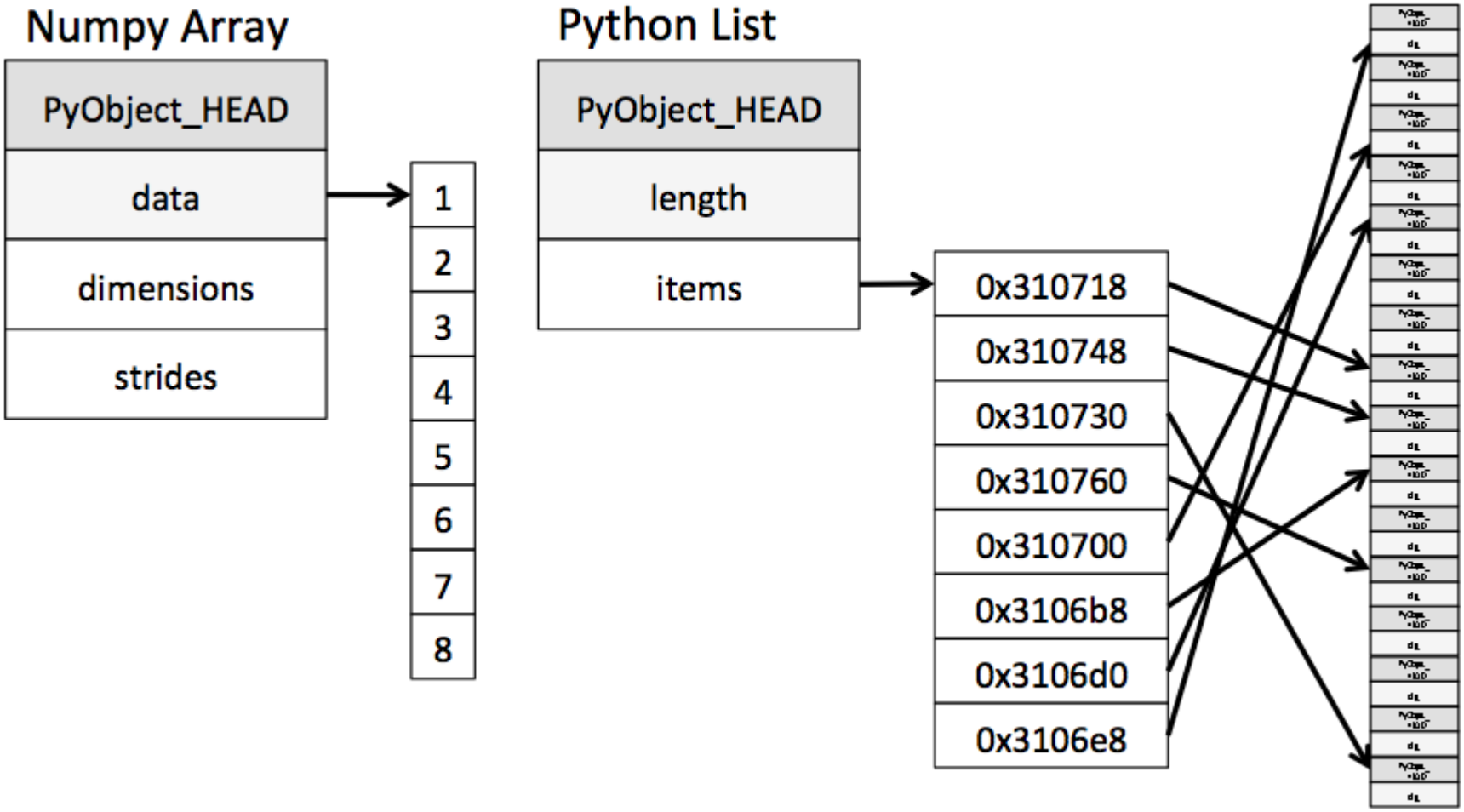
Java keeles annab see koodijupp kompileerimisel vea!

Pythonis ja Javascriptis ei ole üldse tüüpe, kõik muutujad sisaldavad pointereid:

```
x = 34                  # x väärtus on pointer tüübi-info / arvu kombinatsioonile mälus
x =  “John Smith“  # x väärtus on pointer tüübi-info / stringi aadressi kombole
```

Klassikaline massiiv vs Pythoni list

Numpy on Pythoni lisasüsteem kiireks rehkenduseks: tal on karbistamata väärtustega klassikalised massiivid. Javascriptis on samuti olemas „Pythoni moodi“ ja klassikalised.



key-value ehk Pythoni dict ehk Javascripti objekt

Key-value andmestruktuurid otsivad/salvestavad võtme järgi a la :

heights = {“John“: 183, “Mike“: 178, “Greg“: 201}

Ja siis saad küsida või salvestada näiteks

heights[“Mike“]

Kuidas seda teha?

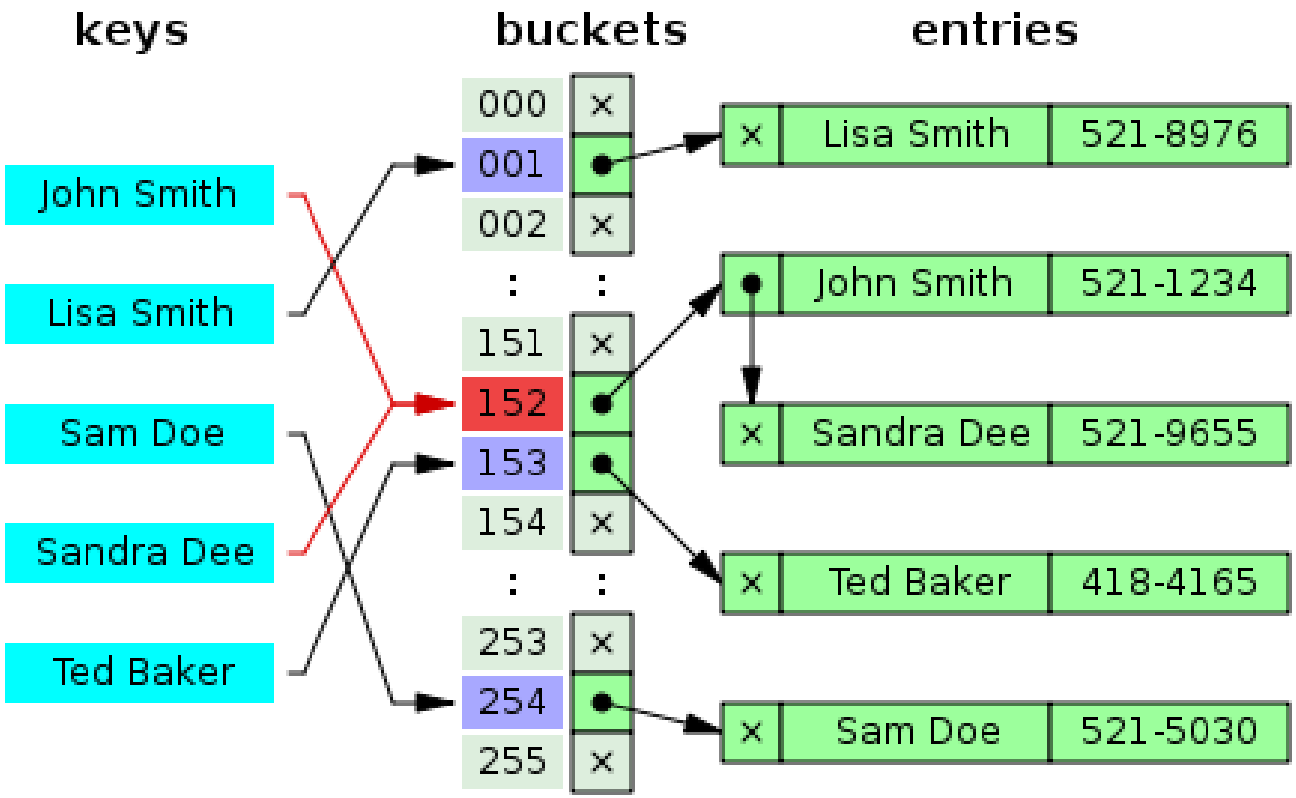
Üks variant oleks teha paaride massiiv või paaride list.

Aga, kui see list läheb pikaks, läheb otsimine aeglaseks.

Hash appi

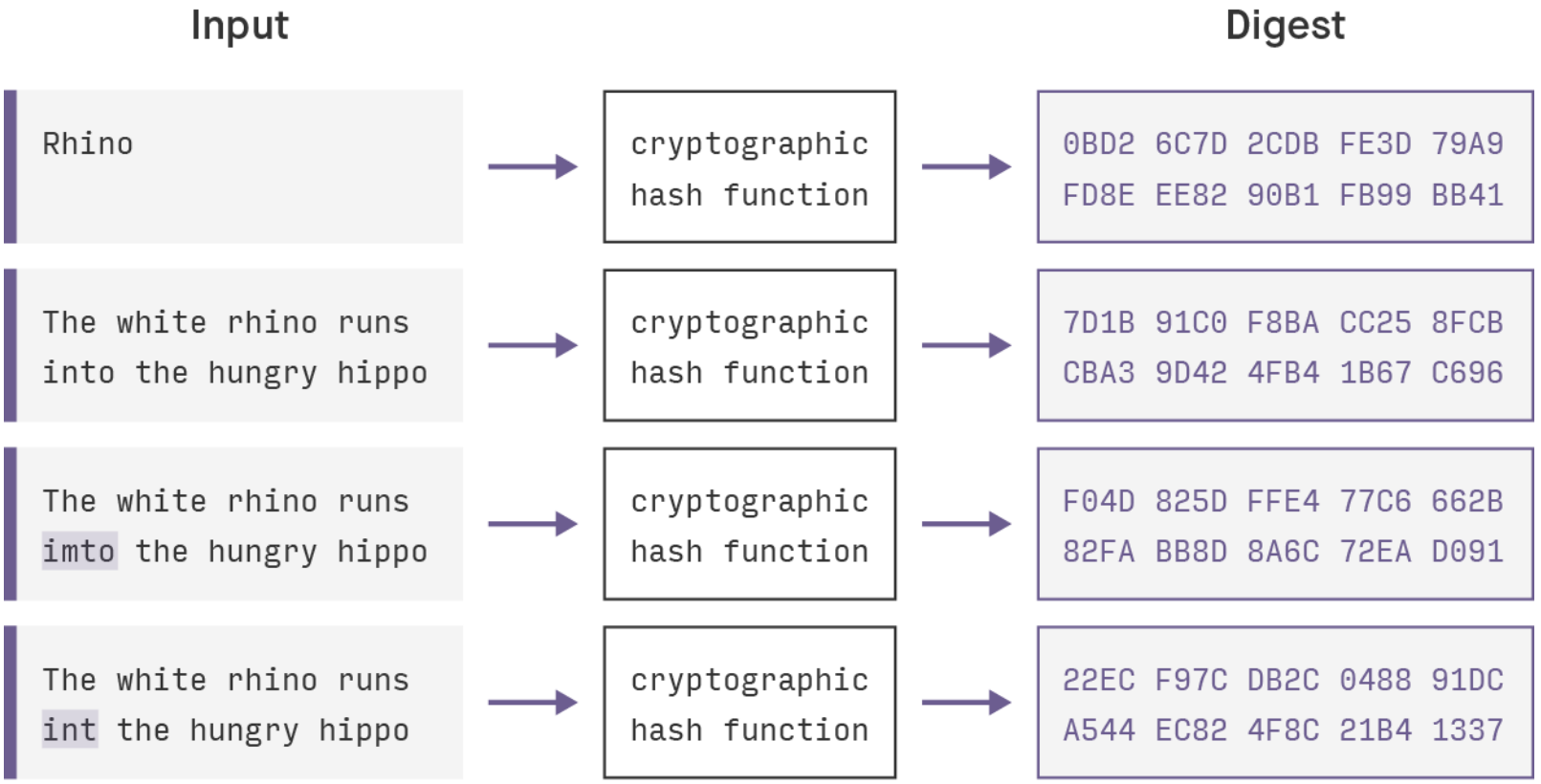
Kiireks leidmiseks ja salvestamiseks kasutatakse **hash tabeleid**.

Python ja Javascript niimoodi oma dicte ja objekte hoiavadki.



Hashi teistmoodi kasutus: cryptohash

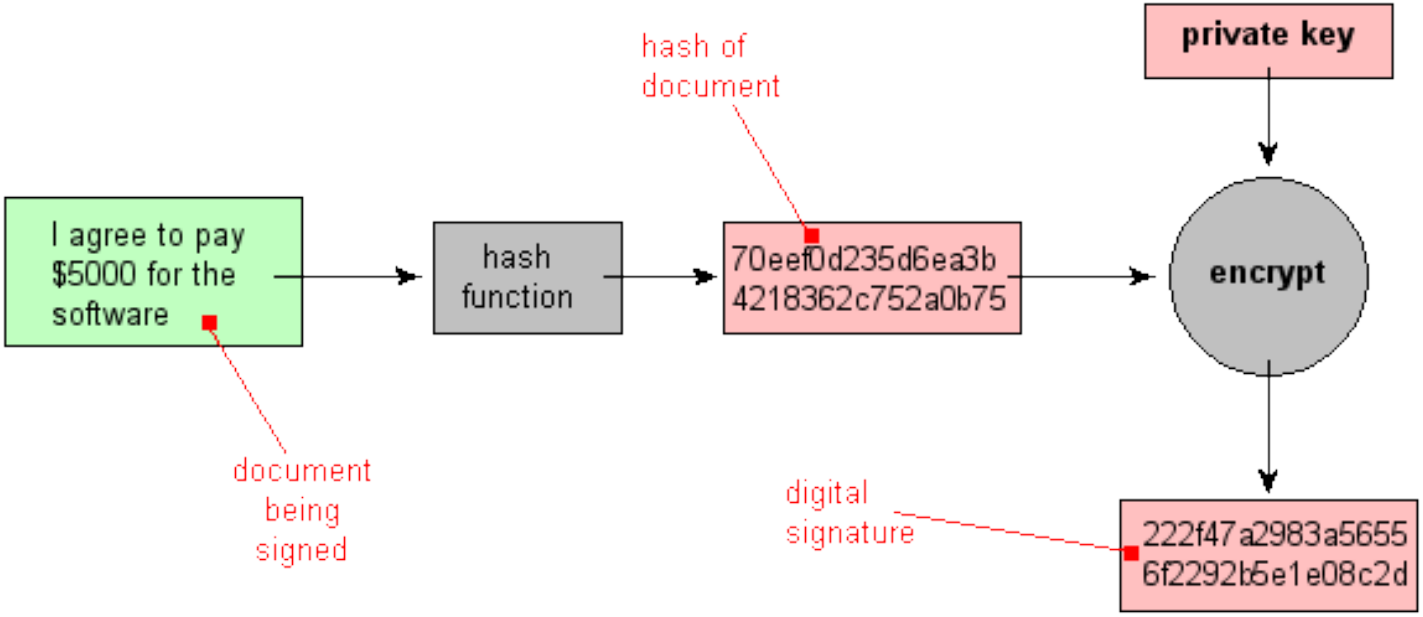
Kui sulle antakse kellegi rehkendatud „digest“, siis sa ei suuda talutava aja jooksul leida või leiutada ühtegi teksti, mis annaks sellesama digesti! Ainus viis leida on proovida järjest kõiki võimalikke tekste ....



Digiallkiri = hash + krüpteering

„Private key“ abil krüpteeritud teksti saab lahti krüpteerida „public key“ abil ja saad kätte hashi ja saad kontrollida, kas oli selle doku hash.

Samas sa ei jaksa leiutada **erinevat dokumenti**, mis annaks sama hashi ja ei jaksa leiutada dokumenti / privaatvõtit, mis annaks sama allkirja.



Tore krüptopuu: Merkle tree

Abiks hulga andmete autentsuse/muutmatuse regulaarseks salvestamiseks.

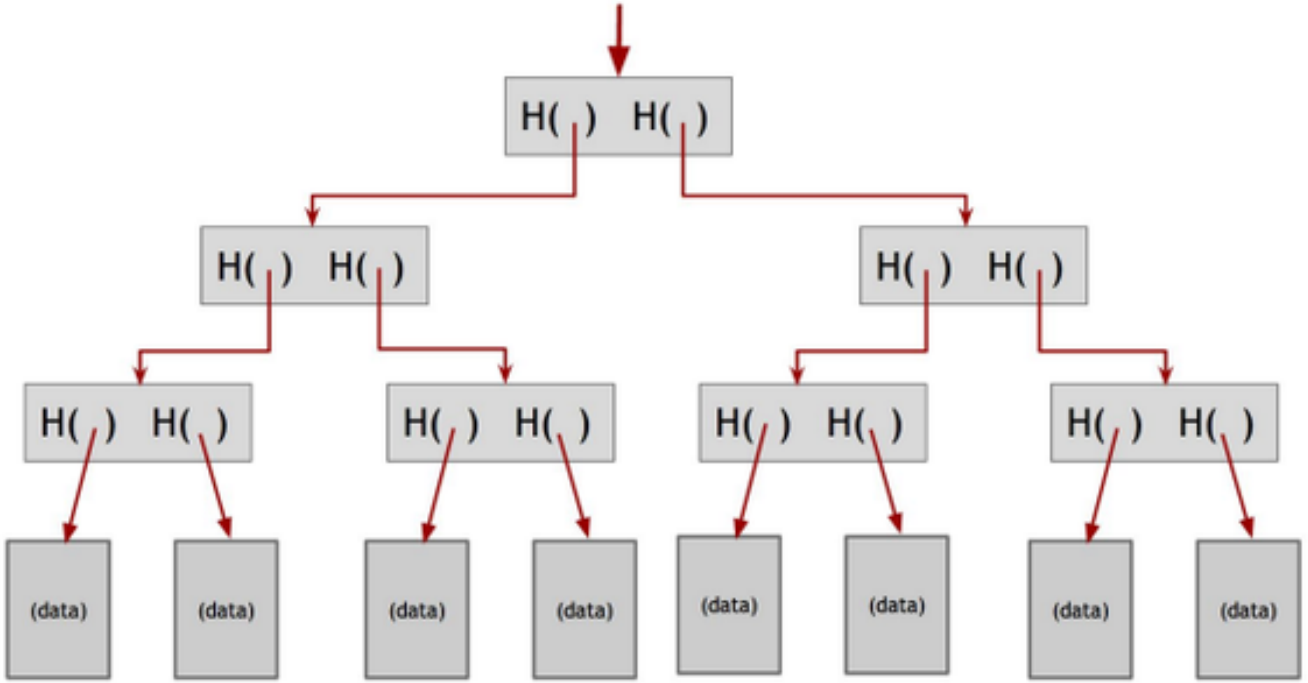


Figure 1.7 Merkle tree. In a Merkle tree, data blocks are grouped in pairs and the hash of each of these blocks is stored in a parent node. The parent nodes are in turn grouped in pairs and their hashes stored one level up the tree. This continues all the way up the tree until we reach the root node.